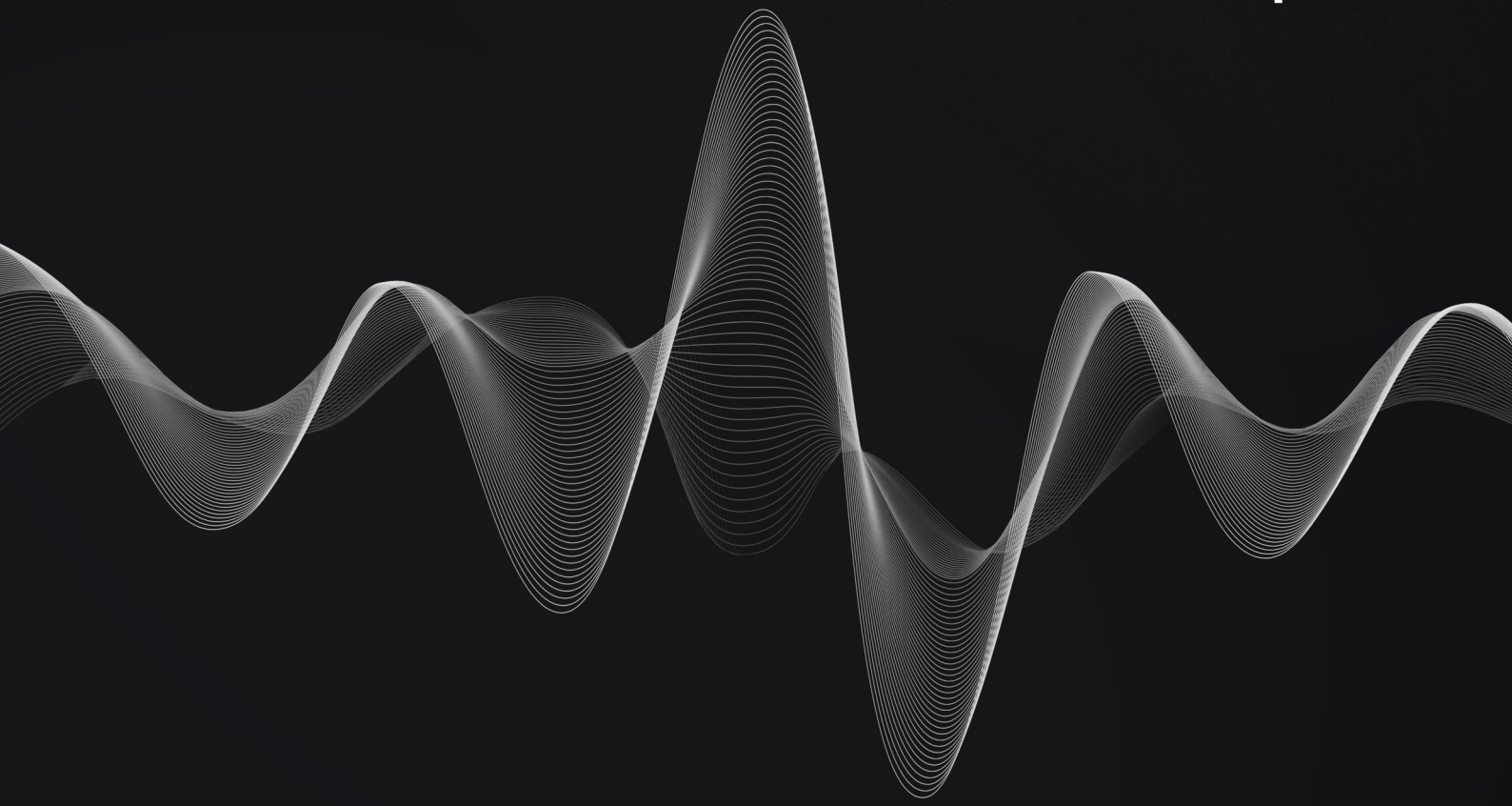
RESONANCE

RTLabs

Demether Restaking Solana Audit Report



Blockchain, Emerging Technology, and Web2
CYBERSECURITY PRODUCT & SERVICE ADVISORY

Document Control

PUBLIC **FINAL**(v2.1)

Audit_Report_DMTH-SOL_FINAL_21

Jan 14, 2025	v0.1	João Simões: Initial draft
Jan 14, 2025	v0.2	João Simões: Added findings
Jan 15, 2025	v1.0	Charles Dray: Approved
Jan 20, 2025	v1.1	João Simões: Reviewed findings
Feb 3, 2025	v2.0	Charles Dray: Finalized
Feb 7, 2025	v2.1	Charles Dray: Published

Points of Contact	Kartik Swaminathan	RTLabs	k@rtlabs.xyz
	Charles Dray	Resonance	charles@resonance.security
Testing Team	João Simões	Resonance	joao@resonance.security
	Michał Bazyli	Resonance	michal@resonance.security
	Luis Arroyo	Resonance	luis.arroyo@resonance.security

Copyright and Disclaimer

Contents

Executive Summary

RTLabs contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between January 8, 2025 and January 15, 2025. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 2 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 5 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide RTLabs with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

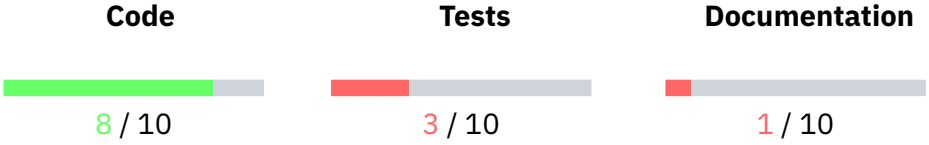
System Overview

Demether is a cutting-edge multichain protocol designed to maximize yield across different blockchain networks. By leveraging a sophisticated blend of restaking, stablecoins, and other financial derivatives, Demether ensures efficient and secure high-yield opportunities for its users. Demether is designed for users seeking to maximize their ETH and SOL yields across multiple blockchain layers while minimizing risk and complexity.

The Demether restaking Solana program implements SPL tokens staking mechanisms into the Jito staking pool and respective vault.

Further bridging functionalities have been added using LayerZero's OFT capabilities, both on EVM-based chains and Solana.

Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is good**.

- Integration tests are included. The tests cover functional requirements. Code coverage is undetermined. Overall, **tests coverage and quality is substandard**.
- The documentation is absent. Overall, **documentation coverage and quality is substandard**.

Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: 4-point-0/demether/programs/demether-restaking/src
- Hash: d62bcce286878b146ea29a3602c8007e7d5afdc2

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

Source Code Review - Rust Solana

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Rust Solana audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Account validation checks
- Account structure and cosplay verifications
- Frontrunning attacks
- Access control issues
- Program-derived addresses and bump seed issues
- Cross-program invocation atomicity and target verifications

- Arithmetic issues
- Denial of service
- Inaccurate business logic implementations
- Anchor-specific vulnerabilities
- Client code interfacing

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

Impact \ Likelihood	Very Likely	Likely	Unlikely
Strong	Critical	High	Medium
Medium	High	Medium	Low
Weak	Medium	Low	Info

Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- **"Quick Win"** Requires little work for a high impact on risk reduction.
- **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

RES-01	Hardcoded Pubkeys Problematic On Immutable Programs	Acknowledged
RES-02	Dead Code	Acknowledged

Hardcoded Pubkeys Problematic On Immutable Programs

Info	RES-DMTH-SOL01	Code Quality	Acknowledged

Code Section

- src/constants.rs

Description

The program specifies accounts that possess validations against constant public keys. These keys cannot be changed unless there is a redeployment of the program into the blockchain. In cases where the accounts related to these public keys become rogue or compromised, the program and users interacting with it may be at risk. This is especially problematic when Solana programs are removed of its update authority and become immutable.

It should be noted that this finding was introduced due to the lack of a deployment script for the program, and as such, should be considered a recommendation for best practices.

Recommendation

It is recommended to ensure that the program is never set as immutable, or must otherwise be redeployed into a new account.

Status

The issue was acknowledged by Demether's team. The development team stated "Validations against constant public keys are to ensure program is only useful for our use case - deposit to Jito stake pool and getting back JitoSOL. Those addresses are unlikely to change, only if Jito decides to make new vault program on new address (to not upgrade current program).".

Dead Code

Info | **RES-DMTH-SOL02** | Code Quality | **Acknowledged**

Code Section

- src/stake_pool_instructions.rs#L82-L84
- src/stake_pool_instructions.rs#L95-L101

Description

Throughout the source code there are multiple instances that lead to a dead end as unimplemented features and functionalities are never reached. These should be cleaned up to increase the quality of the code and decrease the bytecode stored on the blockchain.

Recommendation

It is recommended to clean up the code by removing unnecessary logic that increases the program complexity and decreases code readability, while decreasing the size of the code on the blockchain.

Status

*The issue was acknowledged by Demether's team. The development team stated "Dead code was added in order to be more flexible if upgrading program is needed, we can add option to deposit to stake pool without slippage parameter using current deposit*solix *implementation. Removing that code will almost have no impact on deployment cost because its so small.".*

Proof of Concepts

No Proof-of-Concept was deemed relevant to describe findings in this engagement.